CSE 559A: Computer Vision

[credit: danjodon.deviantart.com]

Fall 2017: T-R: 11:30-1pm @ Lopata 101

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Staff: Abby Stylianou (abby@wustl.edu), Jarett Gross (jarett@wustl.edu)

http://www.cse.wustl.edu/~ayan/courses/cse559a/

Nov 9, 2017

GRADIENT DESCENT

Logistic Regression

For Binary Classification: $\mathcal{X} \rightarrow [0, 1]$

$$f(x; w) = \sigma(w^T \tilde{x}) = \frac{\exp(w^T \tilde{x})}{1 + \exp(w^T \tilde{x})}$$

- Output is interpreted as probability $Pr(y = 1)$
- $w^T \tilde{x}$ are the log-odds.

$$f(x; \theta) = \frac{1}{1 + \exp(-w^T \tilde{x})} \qquad 1 - f(x; \theta) = \frac{1}{1 + \exp(w^T \tilde{x})}$$

$$w^T \tilde{x} = \log \frac{f(x; \theta)}{(1 - f(x; \theta))} = \log \frac{Pr(y = 1)}{1 - Pr(y = 1)} = \log \frac{Pr(y = 1)}{Pr(y = 0)}$$

GRADIENT DESCENT

Logistic Regression

For Binary Classification: $\mathcal{X} \to [0, 1]$

$$f(x; w) = \sigma(w^T \tilde{x}) = \frac{\exp(w^T \tilde{x})}{1 + \exp(w^T \tilde{x})}$$

- To classify, $y = 1$ if $f(x; w) > 0.5$ and 0 otherwise.
- $y = 1$ if $w^T \tilde{x} > 0$, and 0 if $w^T \tilde{x} \leq 0$.
- $\tilde{x}$ is some augmented variable of x.
  - "Linear Classifier" if $\tilde{x} = [x^T; 1]^T$
  - Could be polynomial $\tilde{x} = [1, x, x^2, x^3]$
  - Or other arbitrary non-linear functions of x
- $\tilde{x}$ often called the "feature" vector.
- Some encoding of the input.
  - x could even be non-numeric, and
    you could define some numeric features.

GRADIENT DESCENT

Logistic Regression

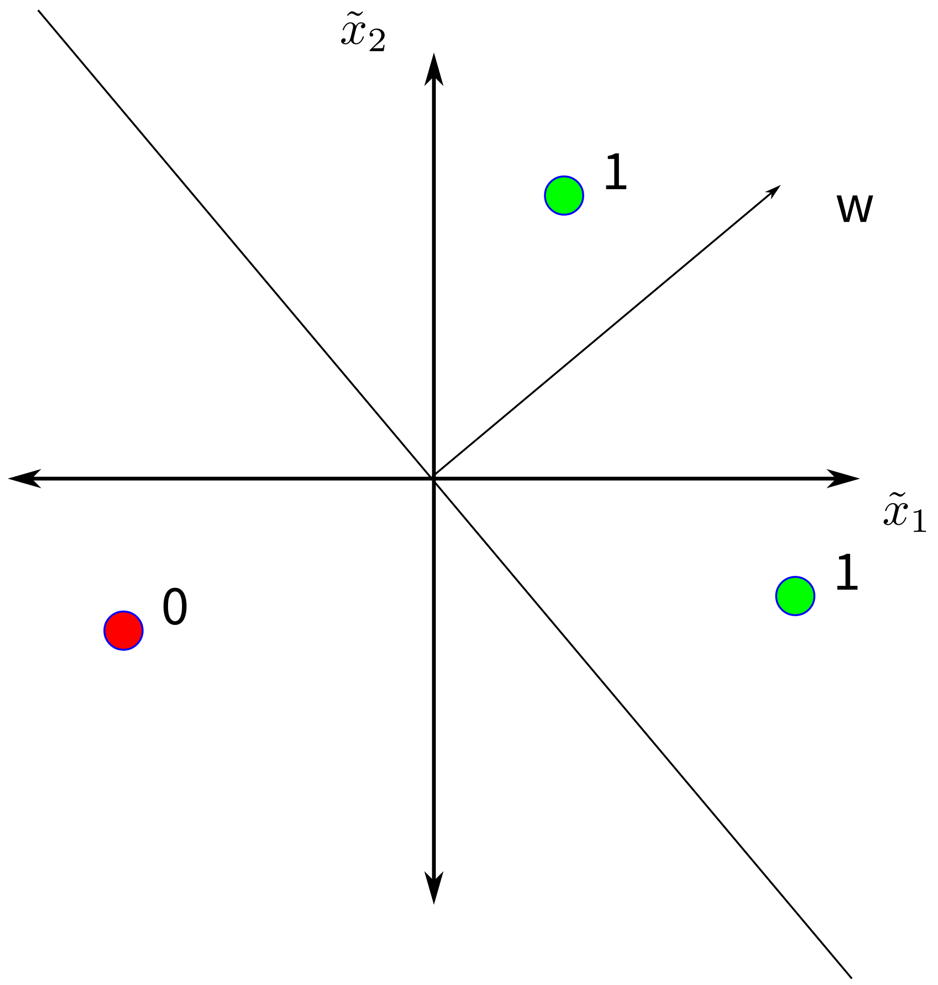
For Binary Classification: $\mathcal{X} \rightarrow [0, 1]$

$$f(x; w) = \sigma(w^T \tilde{x}) = \frac{\exp(w^T \tilde{x})}{1 + \exp(w^T \tilde{x})}$$

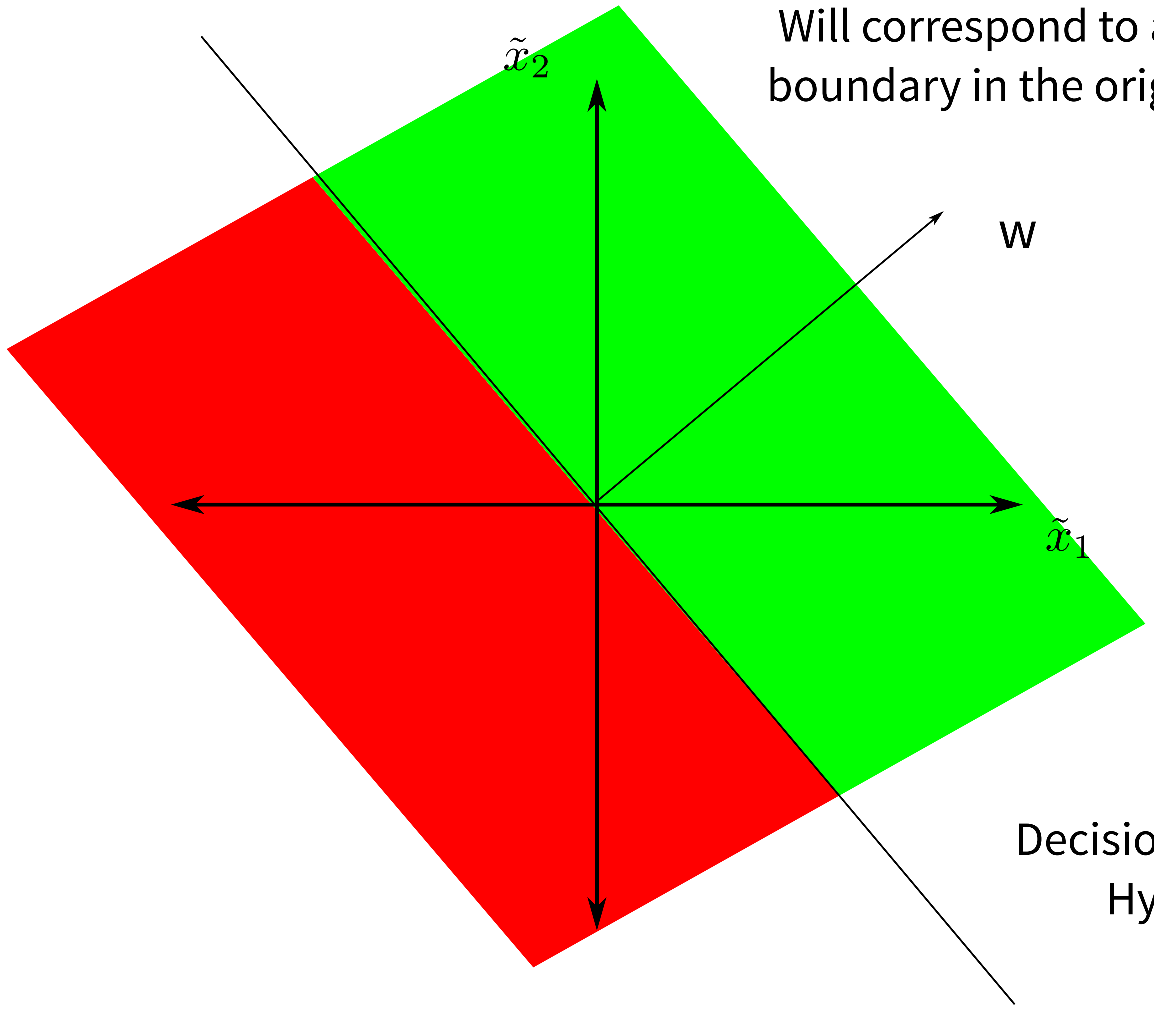
- To classify, $y = 1$ if $w^T \tilde{x} > 0$ and 0 otherwise.
- "Feature engineering" or "Feature selection" was / is often useful
  - E.g., "census transform"
  - Feature vectors based on histograms of gradient orientations (HoG/SIFT)

- Note: Classifier is linear in chosen encoding.
- $w^T \tilde{x} <> 0$ defines a "separating hyperplane" between positive and negative part of the space of $\tilde{x}$.

GRADIENT DESCENT



GRADIENT DESCENT



Will correspond to a non-linear boundary in the original x space

Decision "boundary"
Hyperplane

GRADIENT DESCENT

Logistic Regression

$$f(x;w) = \sigma(w^T\tilde{x}) = \frac{\exp(w^T\tilde{x})}{1+\exp(w^T\tilde{x})}$$

- Cross-entropy / Negative Likelihood Loss

$$L(y, f(x;w)) = -y\log f(x;w) - (1-y)\log(1-f(x;w))$$

$$f(x;\theta) = \frac{1}{1+\exp(-w^T\tilde{x})} \qquad 1-f(x;\theta) = \frac{1}{1+\exp(w^T\tilde{x})}$$

GRADIENT DESCENT

Logistic Regression

$$f(x; w) = \sigma(w^T \tilde{x}) = \frac{\exp(w^T \tilde{x})}{1 + \exp(w^T \tilde{x})}$$

- Cross-entropy / Negative Likelihood Loss

$$L(y, f(x; w)) = y \log\left[1 + \exp(-w^T \tilde{x})\right] + (1 - y) \log\left[1 + \exp(w^T \tilde{x})\right]$$

- Putting it all together, given a training set of $\{(x_t, y_t)\}$:

$$w = \arg\min_{w} \frac{1}{T} \sum_{t=1}^{T} y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

GRADIENT DESCENT

Logistic Regression

$$w = \arg\min_{w} \frac{1}{T} \sum_{t=1}^{T} y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

- You can show that this loss is a convex function of w
  (compute the Hessian matrix and show that it's eigenvalues are non-negative)
- So it has a single global minimum.

But how do we find it ?

GRADIENT DESCENT

Logistic Regression

$$w = \arg\min_w \frac{1}{T} \sum_{t=1}^{T} y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

More General Form

$$w = \arg\min_w C(w) \quad C(w) = \frac{1}{T} \sum_t C_t(w)$$

Minimize a cost that is a function of parameters,
and a **sum** of cost functions, each coming from a different training sample.

Gradient Descent

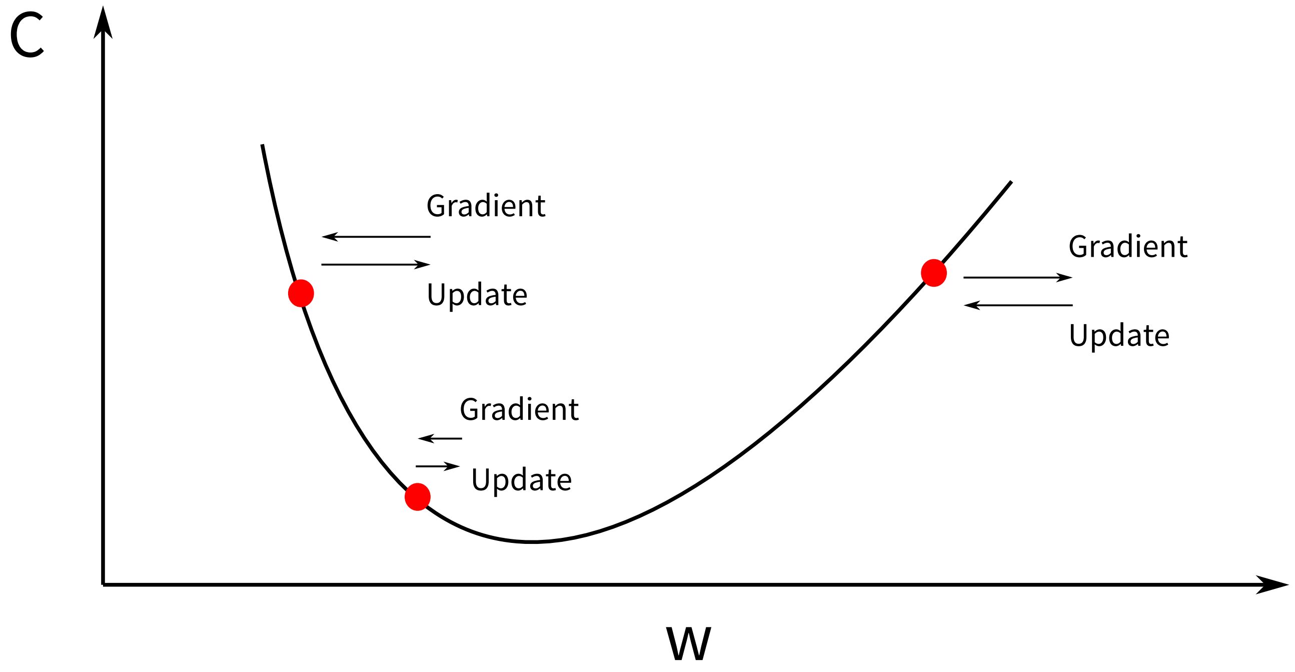
- Begin with initial guess w_0
- At each iteration i:
  - $w_{i+1} \leftarrow w_i - \gamma \nabla_w C(w_i)$

GRADIENT DESCENT

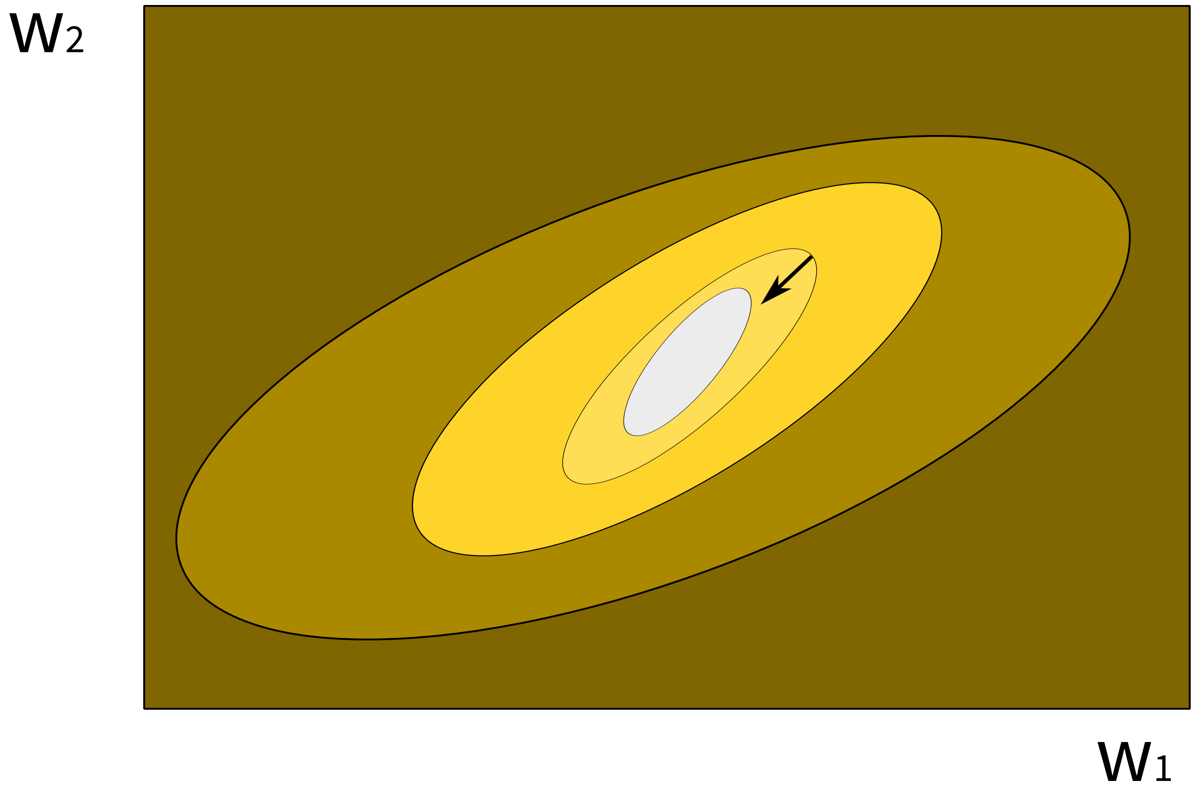
$$w = \arg\min_w C(w) \quad C(w) = \frac{1}{T}\sum_t C_t(w)$$

- Begin with initial guess w_0
- At each iteration i:
  - $w_{i+1} \leftarrow w_i - \gamma \nabla_w C(w_i)$
- At each iteration, we update the parameters w by "moving", in w-space, in the opposite direction of the gradient (at that point w_t).
- We also move by length proportional to the magnitude of the gradient.
- γ is the step-size. When running optimization for training, often called the "learning rate".

GRADIENT DESCENT



GRADIENT DESCENT



GRADIENT DESCENT

- If you select optimal step size by doing a "line search" for γ, can prove that gradient-descent will converge.
- If function is convex, converge to unique global minimum.
- Second order variants that consider the Hessian matrix: Newton & Quasi-Newton Methods
    - Gauss-Newton, Levenberg-Marquardt, ...

But simple gradient descent suffices / our only choice when:

- Function isn't convex.
- Can't afford to do line search.
- So many parameters that can't compute Hessian.

Also, no theoretical guarantees.

Theory still catching up. Meanwhile, we'll try to understand the "behavior" of the gradients.

GRADIENT DESCENT

$$\nabla_w C(w) = \begin{bmatrix} \frac{\partial}{\partial w_1} C(w) \\ \frac{\partial}{\partial w_2} C(w) \\ \vdots \end{bmatrix}$$

$$\text{If} \quad C(w) = \frac{1}{T} \sum_t C_t(w), \quad \text{then} \quad \nabla_w C(w) = \frac{1}{T} \nabla_w C_t(w)$$

Logistic Regression

$$C_t(w) = y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

What is $\nabla_w C_t(w)$, the gradient of the loss from a singe training example ?

GRADIENT DESCENT

$$C_t(w) = y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

Ok, what is the derivative of

$$C_t(p) = y_t \log\left[1 + \exp(-p)\right] + (1 - y_t) \log\left[1 + \exp(p)\right]$$

with respect to p (where p is a scalar).

$$\frac{\partial}{\partial p} C_t(p) = \frac{\exp(p)}{1 + \exp(p)} - y_t$$

$$\frac{\partial}{\partial p} C_t(p) = y_t \quad \frac{-\exp(-p)}{1 + \exp(-p)} + (1 - y_t) \frac{\exp(p)}{1 + \exp(p)}$$

$$= \frac{\exp(p)}{1 + \exp(p)} - y_t \left[\frac{\exp(-p)}{1 + \exp(-p)} + \frac{\exp(p)}{1 + \exp(p)}\right]$$

$$= \frac{\exp(p)}{1 + \exp(p)} - y_t \left[\frac{1}{1 + \exp(p)} + \frac{\exp(p)}{1 + \exp(p)}\right]$$

GRADIENT DESCENT

$$C_t(w) = y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

$$C_t(p) = y_t \log\left[1 + \exp(-p)\right] + (1 - y_t) \log\left[1 + \exp(p)\right]$$

$$\frac{\partial}{\partial p} C_t(p) = \frac{\exp(p)}{1 + \exp(p)} - y_t$$

Observations

- $\frac{\exp(p)}{1+\exp(p)}$ is basically the output $f(x_t; w)$, predicted probability that $y_t = 1$.
- Remember: this is the expression for gradient of p, i.e. logit / log-odds.
- Gradient 0 if $y_t = 0$ and probability 0, $y = 1$ and probability 1.
    - Do nothing if predicting right answer with perfect confidence.
- If we say probability > 0, and $y_t = 0$. Gradient is positive.
- If we say probability < 1, and $y_t = 1$. Gradient is negative.

Remember we move in the opposite direction of gradient.

GRADIENT DESCENT

$$C_t(w) = y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

$$C_t(p) = y_t \log\left[1 + \exp(-p)\right] + (1 - y_t) \log\left[1 + \exp(p)\right]$$
$$\frac{\partial}{\partial p} C_t(p) = \frac{\exp(p)}{1 + \exp(p)} - y_t$$

Also, changing p makes a much bigger difference in the corresponding probability, when p is near 0 / probability near 0.5.

GRADIENT DESCENT

$$C_t(w) = y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

$$C_t(p) = y_t \log\left[1 + \exp(-p)\right] + (1 - y_t) \log\left[1 + \exp(p)\right]$$

$$\frac{\partial}{\partial p} C_t(p) = \frac{\exp(p)}{1 + \exp(p)} - y_t$$

But this is still derivative with respect to p. We want gradient with respect to w.

$$\frac{\partial}{\partial w^j} C_t(w) = \tilde{x}^j \quad \times \quad \left[\frac{\exp(p)}{1 + \exp(w^T \tilde{x}_t)} - y_t\right]$$

$$\nabla_w C_t(w) = \tilde{x} \quad \left[\frac{\exp(p)}{1 + \exp(w^T \tilde{x}_t)} - y_t\right]$$

$$\nabla_w C_t(w) = \nabla_w (w^T \tilde{x}) \quad \left[\frac{\exp(p)}{1 + \exp(w^T \tilde{x}_t)} - y_t\right]$$

$$\nabla_w C_t(w) = \nabla_w p(w) \quad \frac{\partial C_t(p)}{\partial p}$$

GRADIENT DESCENT

$$w = \arg\min_w \frac{1}{T} \sum_{i=1}^{T} y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

Putting it together:

- At each iteration i,
  - Based on current w, compute $f(x_t, w) = \hat{y}_t$
  - Compute derivative of the "output" as $\hat{y}_t - y_t$
  - Multiply by x_t to get ∇_w
  - Change w by subtracting some γ times this gradient.

GRADIENT DESCENT

$$w = \arg\min_{w} \frac{1}{T} \sum_{i=1}^{T} y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

Putting it together:

- At each iteration i,
  - Based on current w, compute $f(x_t, w) = \hat{y}_t$ for every training sample
  - Compute derivative of the "output" as $\hat{y}_t - y_t$ for every training sample
  - Multiply by x_t and average across all training samples to get ∇_w
  - Change w by subtracting some γ times this gradient.

Expensive when we have a LOT of training data.

STOCHASTIC GRADIENT DESCENT

$$w = \arg\min_w \frac{1}{T} \sum_t C(x_t, y_t; w)$$

$$\nabla_w = \frac{1}{T} \sum_t \nabla_w C(x_t, y_t; w)$$

Remember, summation over training samples meant to approximate an expectation over $P_{XY}(x, y)$.

$$\frac{1}{T} \sum_t C(x_t, y_t; w) \rightarrow \mathbb{E}_{P_{XY}(x,y)} C(x, y; w)$$

$$\frac{1}{T} \sum_t \nabla_w C(x_t, y_t; w) \rightarrow \mathbb{E}_{P_{XY}(x,y)} \nabla_w C(x, y; w)$$

In other words, we are approximating the "true" gradient with gradients over samples.

What if we used a smaller number of samples in each iteration, but different samples in different iterations ?

STOCHASTIC GRADIENT DESCENT

- Single sample
$$w_{i+1} \leftarrow w_i - \gamma \nabla_w C_t(x_t, y_t; w_i)$$
At each iteration, choose a random $t \in \{1, 2, \dots, T\}$.
- "Mini"-batched SGD (sometimes GD is called Batched GD)
$$w_{i+1} \leftarrow w_i - \gamma \nabla_w \frac{1}{B} \sum_{t \in \mathcal{B}} C_t(x_t, y_t; w_i)$$
At each iteration, choose a random smaller batch $\mathcal{B}$ of size $B << T$.

With replacement ? Without replacement ?

STOCHASTIC GRADIENT DESCENT

In practice:

- Shuffle order of training examples
- Choose a batch size
- Take consecutive groups of B samples as you loop through iterations
  - [1,B] in iteration 1
  - [B+1,2B] in iteration 2
  - . . .
- Once you reach the end of the training set (called one "epoch"), shuffle the order again.

STOCHASTIC GRADIENT DESCENT

$$w_{i+1} \leftarrow w_i - \gamma \frac{1}{B} \sum_{t \in \mathcal{B}} \nabla_w C_t(x_t, y_t; w_i)$$

General Notes

- The gradient over a mini-batch is an "approximation", or a "noisy" version of the gradient over the true training set.

$$\frac{1}{B} \sum_{t \in \mathcal{B}} \nabla_w C_t(x_t, y_t; w_i) = \frac{1}{T} \sum_{t=1}^{T} \nabla_w C_t(x_t, y_t; w_i) + \epsilon$$

- Typically, if you decrease the batch-size, you will want to decrease your step size (because you are "less sure" about the gradient).

STOCHASTIC GRADIENT DESCENT

$$w_{i+1} \leftarrow w_i - \gamma \frac{1}{B} \sum_{t \in \mathcal{B}} \nabla_w C_t(x_t, y_t; w_i)$$

General Notes

Say your cost function is convex, and you care only about decreasing this cost (not worried about overfitting)

- Larger batch size will always give you "better" gradients.
- But diminishing returns after a batch size.
- Computational cost is number of examples per iteration $\times$ number of iterations for convergence
  - Higher batch means more computation per iteration, but may mean fewer iterations required to converge.
- Best combination of step size and batch size is an empirical question.
- Another factor: parallelism.
  - Note that you can compute the gradient of all samples of your batch in parallel.
  - Ideally, you want to at least "saturate" all available parallel threads.

STOCHASTIC GRADIENT DESCENT

$$w_{i+1} \leftarrow w_i - \gamma \frac{1}{B} \sum_{t \in \mathcal{B}} \nabla_w C_t(x_t, y_t; w_i)$$

General Notes

If your cost function is NOT convex, and/or you are worried about overfitting.

- Noise in your gradients might be a good thing !
- Might help you escape local minima.
- Might prevent you from overfitting to train set.
- Try different batch sizes, check performance on dev set, not just train set.

STOCHASTIC GRADIENT DESCENT

Momentum

Standard SGD:

$$g_{i+1} = \frac{1}{B} \sum_{t \in \mathcal{B}} \nabla_w C_t(x_t, y_t; w_i)$$
$$w_{i+1} \leftarrow w_i - \gamma g_{i+1}$$

With Momentum:

For $\beta < 1$:

$$g_{i+1} = \frac{1}{B} \sum_{t \in \mathcal{B}} \nabla_w C_t(x_t, y_t; w_i) + \beta g_i$$
$$w_{i+1} \leftarrow w_i - \gamma g_{i+1}$$

- Keep adding the gradient from a previous batch, again and again across iterations, with decaying weight.
- Remember: g_i was computed with respect to a different position in w space.
- People often use β as high as 0.9 or 0.99.
- Will need to revisit "best" value of γ when you change β.

LOGISTIC REGRESSION

$$w = \arg\min_w \frac{1}{T} \sum_t C_t(w)$$

$$C_t(w) = y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

- Defined linear classifier on augmented vector $\tilde{x}$
- Used gradient descent to learn w.
  - Looked at behavior of gradients.
  - Simplified computation with stochasticity.
- At test time, sign of $w^T \tilde{x}$ gives us our label.

The problem is:

- The definition of augmented vector $\tilde{x}$ is hand-crafted
- We have manually engineered our features.
- The only thing we're learning is a linear classifier on top.

Want to learn the features themselves !

Given that SGD works, what's stopping us from learning a $g(x) = \tilde{x}$?